UNIVERSIDADE DE ÉVORA

Agenda
Estrutura de Dados e Algoritmos I

Relatório Técnico

Tomás Rato nº62755
Diogo Marona nº63164
Luís Canhoto nº63358
João Padeiro nº63655

Junho
Ano Letivo 2024/2025

Docente: Prof.ª Lígia Ferreira

Conteúdo

1 Introdução **3**

2 Estrutura de Dados Utilizadas **4**
2.1 Gestão de Contactos - Tabela de Hash com encadeamento 4
2.2 Pesquisa de Contactos - ABP 4
2.3 Identificação do Chamador - AVL 4
2.4 Histórico de Chamadas - Lista Duplamente Ligada Circular 4

3 Descrição Funcional **5**
3.1 Inserção de Contactos 5
3.2 Edição e Remoção de Contactos 5
3.3 Pesquisa de Contactos 5
3.4 Listagem de Contactos por Ordem Alfabética 5
3.5 Identificação do Chamador 6
3.6 Registo de Chamadas 6
3.7 Navegação no Histórico 6

4 Organização do Código **7**
4.1 Estrutura Geral 7
4.2 Ficheiro main.c 7
4.3 Gestão de Contactos - hash.c/hash.h 8
4.4 Pesquisa de Contactos - abp.c/abp.h 8
4.5 Identificação do Chamador - avl.c/avl.h 8
4.6 Histórico de Chamadas - doublelist.c/doublelist.h 9
4.7 Manipulação de Ficheiros - ficheiros.c/ficheiros.h 9
4.8 Estrutura de Dados Base - contacto.h 9
4.9 Interface Visual - cores.h 9

5 Dificuldades Encontradas **10**
5.1 Integração de Múltiplas Estrutuas de Dados 10
5.2 Gestão de Colisões na Tabela de Hash 10
5.3 Pesquisa Parcial na ABP 10
5.4 Histórico de Chamadas com Navegação Interativa 11
5.5 Leitura e Escrita em Ficheiros 11
5.6 Normalização de Strings 11

6 Funcionalidades Extra **12**
6.1 Persistência de Dados com Ficheiros 12
6.2 Interface com Cores no Terminal 12

7 Compilação e Execução **13**
7.1 Requisitos 13
7.2 Compilação 13
7.3 Execução 13
7.4 Ficheiros Gerados 13
7.5 Notas 13

8 Conclusão **14**

1 Introdução

Este projeto tem como objetivo o desenvolvimento de uma aplicação em linguagem C que simula uma agenda de contactos com registo de chamadas, integrando diferentes estruturas de dados estudadas na unidade curricular de **Estrutura de Dados e Algoritmos I**.

A aplicação permite ao utilizador gerir contactos com alguns comandos (como adicionar, remover, editar, entre outros), bem como manter um histórico de chamadas organizadas por ordem cronológica. Para isso, foram utilizadas estruturas de dados adequadas garantindo uma separação clara entre os módulos da aplicação.

Todo o sistema foi desenvolvido com base na modularização do código, recorrendo à criação de TADs (Tipos Abstratos de Dados) próprios e organizando o projeto em ficheiros de cabeçalho (.h) e implementação (.c), de modo a facilitar a manuntenção e leitura do código. Este relatório descreve as decisões tomadas ao longo do desenvolvimento, as estruturas implementadas e os principais desafios encontrados.

2 Estrutura de Dados Utilizadas

Como já referido, a aplicação foi desenvolvida com base em várias estruturas de dados, escolhidas de orma a otimizar o desempenho das funcionalidades principais e respeitar a redtrição de não reutilizar a mesma estrutura para finalidades distintas. Cada estrutura foi implementada de forma modular, adaptada às necessidades específicas do sistema.

2.1 Gestão de Contactos - Tabela de Hash com encadeamento

Para a gestão de contactos, foi utilizada uma Tabela de Hash com encadeamento, permitindo acesso rápido aos dados através do nome do contacto. Esta estrutura garante uma complexidade próxima de **O(1)** nas operações de inserções, remoção e atualização, tornando-se ideal para um grande volume de contactos pois assegura o correto armazenamento e recuperação dos dados.

2.2 Pesquisa de Contactos - ABP

A pesquisa de contactos por nomes completos ou parciais foi implementada através de uma ABP. Esta estrutura permite percorrer os elementos por ordem alfabética. Além disso, possibilita a pesquisa eficiente com complexidade, em média, **O(log(n))** quando a árvore está equilibrada, mantendo o desempenho aceitável mesmo em grandes conjuntos de dados.

2.3 Identificação do Chamador - AVL

Para a identificaçãp do chamador, foi utilizada uma Árvore AVL, uma variação balanceada da ABP. A AVL mantém a altura da árvore sempre equilibrada, garantindo tempos de acesso e inserção de **O(log(n))** no pior caso. Esta estrutura foi usada para associar números de telemóvel a nomes, permitindo uma resposta rápida nos três cenários definidos: número oculto, número conhecido e número desconhecido.

2.4 Histórico de Chamadas - Lista Duplamente Ligada Circular

O histórico de chamadas foi gerido com uma lista duplamente ligada circular, onde as chamadas mais recentes são adicionadas ao início da lista. Esta estrutura permite a navegação bidirecional no histórico, conforme exigido, e facilita a implementação de funcionalidades mantendo a simplicidade da lógica e a clareza na gestão dos elementos.

3 Descrição Funcional

A aplicação desenvolvida simula uma agenda de contactos com funcionalidades de gestão, pesquisa e histórico de chamadas. Foi construída com várias estruturas de dados, cada uma responsável por um tipo de operação, garantindo modularidade e eficiência. Este tópico descreve as funcionalidades implementadas.

3.1 Inserção de Contactos

A criação de novos contactos é feita através da função ***Insert()***, que insere o contacto na tabela de hash. Para garantir nomes únicos, é utilizado o mecanismo ***CreateUniqueName()*** que acrescenta um sufixo ao nome caso este já exista. O contacto inserido na hash é posteriormente replicado na árvore ABP (para pesquisa) e na AVL (para identificação de chamadas).

3.2 Edição e Remoção de Contactos

A edição de um contacto permite alterar o nome e/ou número. A alteração é feita diretamente no concacto obtido por ***Find()***, garantindo coerência na hash. A função de remoção percorre a lista ligada da posição da tabela de hash correspondente ao nome e remove o contacto. Após estas alterações, as árvores ABP e AVL são reconstruídas (através de ***CreateTree()*** e ***CreateAVL()***).

3.3 Pesquisa de Contactos

A pesquisa é feita através da ABP, que é construída a partir dos dados da tabela de hash. A função ***FindABP()*** permite pesquisar termos parciais utilizando ***Normalizar()*** para garantir que letras acentuadas e maiúsculas são tratadas uniformemente. A pesquisa percorre a árvore por completo, imprimindo todos os nomes que contenham o termo procurado.

3.4 Listagem de Contactos por Ordem Alfabética

A função ***PrintTable()*** da hash gera uma cópia dos contactos, ordena-os alfabeticamente (usando a função Normalizar para comparação e um sistema de ordenação SelectionSort) e imprime-os. Embora a hash não esteja ordenada internamente, esta abordagem permite listar os contactos por ordem alfabética sem depender da ABP.

3.5 Identificação do Chamador

A função ***IdentifyCaller()*** identifica o chamador de uma chamada com base em três casos:

- Se o número for "OCULTO", o nome apresentado é "UNKNOWN".
- Se o número estiver presente na AVL (criada de acordo com os números de telmóvel), o nome correspondente é devolvido.
- Caso contrário, é devolvido o próprio número como identificador

Esta função é usada na simulação de chamadas para preencher o campo "chamador"da estrutura "Chamada".

3.6 Registo de Chamadas

As chamadas são representadas por uma estrutura Chamada e registadas numa lista duplamente ligada circular, através da função ***InsertList()***. Cada chamada contém o nome do chamador, destinatário, número e duração. As chamadas são inseridas no início da lista, mantendo a ordem cronológica inversa.

3.7 Navegação no Histórico

A navegação do histórico de chamadas é interativa e feita com base num cursor que percorre a lista duplamente ligada circular. A função ***NavegarHistorico()*** permite ao utilizador visualizar uma chamada de cada vez e navegar para a próxima ou anterior usando comandos no teclado. Como a lista é circular, ao chegar ao fim reinicia-se no início e vice-versa, assegurando uma experiência contínua na navegação.

4 Organização do Código

O projeto foi desenvolvido em linguagem C, de forma modular, com separação clara entre ficheiros de cabeçalho (.h) e de implementação (.c). Cada componente do sistema (contactos, pesquisa, identificação, chamadas e histórico) foi isolado num conjunto próprio de ficheiros, respeitando os princípios de reutilização.

4.1 Estrutura Geral

A organização dos módulos é a seguinte:

Ficheiros	Função Principal
main.c	Ponto de entrada do programa. Inicializa estruturas, carrega dados e chama o menu principal.
programa.c/programa.h	Interface com o utilizador. Contém o menu principal e a ligação entre os módulos.
hash.c/hash.h	Gestão de contactos com Tabela de Hash com encadeamento externo para a gestão de contactos.
abp.c/abp.h	Pesquisa de contactos por nome ou parcial com ABP.
avl.c/avl.h	Identificação do chamador com AVL (com base no número).
chamada.c/chamada.h	Define a estrutura "Chamada"e a lógica de identifiação do chamador.
doublelist.c/doublelist.h	Lista duplamente ligada circular para histórico de chamadas. Possui navegação interativa.
ficheiros.c/ficheiros.h	Carregamento e gravação de contactos e histórico a partir de ficheiros de texto.
contacto.h	Define a estrutura base do contacto (nome e número) usada em toda a aplicação.
cores.h	Defne códigos ANSI para cores no terminal, usados na interface textual.

4.2 Ficheiro main.c

Este ficheiro contém a função ***main()***, que é o ponto de entrada do programa. A sua responsabilidade é inicializar as estruturas de dados principais (a tabela de hash e a lista duplamente ligada), carregar os dados guardados em ficheiros, executar o menu interativo e, no fim, guardar os dados atualizados e libertar a memória alocada. As principais chamadas são:

- InitializeTable() e CreateList();
- LoadContactos() e LoadHistorico();
- MenuInicial();
- SaveContactos() e SaveHistorico();
- DestroyTable() e DestroyList().

4.3 Gestão de Contactos - hash.c/hash.h

Este módulo implementa uma Tabela de Hash com encadeamento externo (listas ligadas), usada como estrutura central para armazenar os contactos. Permite inserção, remoção, pesquisa direta e listagem ordenada. As suas funções são:

- Insert(), Delete(), Find(), CreateUniqueName();
- PrintTable() (com ordenação tipo bubble sort);
- Normalizar() (remoção de acentos e conversão para minúsculas);
- InitializeTable(), DestroyTable().

4.4 Pesquisa de Contactos - abp.c/abp.h

Este módulo constrói uma ABP com base nos contactos da hash. A árvore é usada para pesquisar contactos por nome completo ou parcial. As suas funções são:

- CreateTree() - cria a ABP a partir da hash;
- InsertABP(), MakeABPEmpty();
- FindABP() - percorre toda a árvore e imprime nomes que contenham o termo pesquisado.

4.5 Identificação do Chamador - avl.c/avl.h

A identificação do chamador durante uma chamada é feita com uma AVL, construída com os contactos e organizada por número de telemóvel. As suas funções são:

- CreateAVL(), InsertAVL(), MakeAVLEmpty();
- FindByNumero() - pesquisa eficiente do número de telemóvel;
- IdentifyCaller() - lógica para determinar se o chamador é conhecido, oculto ou desconhecido.

4.6 Histórico de Chamadas - doublelist.c/doublelist.h

Este módulo implementa uma lista duplamente ligada ciruclar, usada para armazenar o historico de chamadas. As chamadas mais recentes são inseridas no início da lista. Temos as seguintes funcionalidades:

- InsertList() - insere a chamada na lista;
- InsertByFicheiros() - insere chamadas a partir do ficheiro (no final);
- PrintList() - inprime todas as chamadas
- NavegarHistorico() - interface interativa para navegar para frente/trás;
- Avancar() e Retroceder() - encapsulam a navegação circular;
- DestroyList().

4.7 Manipulação de Ficheiros - ficheiros.c/ficheiros.h

Este módulo trata da persistência dos dados entre sessões. Os contactos são guardados em "CONTACTOS.TXT"e o histórico em "HISTORICO.txt". Eis as funções:

- SaveContactos()/LoadContactos();
- SaveHistorico()/LoadHistorico().

4.8 Estrutura de Dados Base - contacto.h

Define a estrutura "Contacto", composta por nome e número de telemóvel. É usada por quase todos os módulos como elemento base.

4.9 Interface Visual - cores.h

Define macros para códigos ANSI que permitem imprimir texto colorido no terminal. Utilizado para destacar menus, mensagens e resultados.

5 Dificuldades Encontradas

Durante o desenvolvimento do projeto, surgiram várias dificuldades técnicas e estruturais que exigiram soluções criativas e uma compreensão aprofundada das estruturas de dados envolvidas.

5.1 Integração de Múltiplas Estrutuas de Dados

Uma das maiores dificuldades foi garantir a consistência entre todas as estruturas (Hash, ABP, AVL, Lista) sempre que um contacto era inserido, editado ou removido. Como estas estruturas são reconstruídas a partir da hash, era importante evitar redundância e manter os dados sincronizados.

Foi necessário definir bem quando reconstruir as árvores e quando operar diretamente sobre a hash para não comprometer a integridade dos dados.

5.2 Gestão de Colisões na Tabela de Hash

A implementação da tabela de hash com encadeamento externo foi relativamente simples no que toca à inserção, mas exigiu cuidado na remoção e procura, principalmente ao garantir que nomes duplicados não colidissem logicamente. Para isso, foi criada a função CreateUniqueName(), que gera nomes únicos automaticamente - uma solução simples mas eficaz.

A tabela de hash tem tamanho fixo de 1009 pois é um número primo alto o que evita um maior número de colisões nas inserções. Ao criar a chave, na função hash, foi utilizado um método de multiplicar 37 (número primo ímpar) pelo valor já acumulado e somar o valor ASCII do caracter lido. Este tipo de função dispersora gera uma distribuição mais uniforme das chaves, contribuindo para a eficiência da hash.

Apesar da tabela de hash já permitir, basicamente, espaço ilimitado, ao recorrermos as estas técnicas conseguimos uma melhor gestão da memória utilizada do programa e operações de inserção e procura mais fluídas e naturais.

5.3 Pesquisa Parcial na ABP

Outra dificuldade foi implementar uma pesquisa de nomes que não usasse apenas comparação binária direta, mas permitisse encontrar termos contidos dentro dos nomes (ex: pesquisar "ana"encontra "Mariana"). Isso obrigou-nos a fazer um percurso completo da ABP com normalização de strings, perdendo a eficiência da ABP tradicional mas ganhando flexibilidade funcional.

5.4 Histórico de Chamadas com Navegação Interativa

A gestão do histórico com uma lista duplamente ligada circular foi bem-sucedida, mas a implementação da navegação interativa exigiu especial atenção. Foi necessário criar um cursor e controlar o avanço e retrocesso, tratando também os casos em que se chega ao fim ou ao início da lista - sem criar ciclos infinitos ou falhas no input.

5.5 Leitura e Escrita em Ficheiros

A leitura dos dados a partir de ficheiros implicou lidar com formatações específicas (como separadores ";"), bem como garantir que os dados carregadores fossem inseridos corretamente nas estruturas. Para manter o histórico na ordem correta, foi criada uma função separada (InsertByFicheiros()) que insere chamadas no final da lista, preservando a ordem cronológica inversa.

5.6 Normalização de Strings

A necessidade de comparar nomes independentemente de maiúsculas/minúsculas ou acentuação exigiu o desenvolvimento de uma função de normalização (Normalizar()), que remove acentos e converte tudo para minúsculas. Esta função exigiu um mapeamento manual de caracteres UTF-8 comuns e foi fundamental para a pesquisa e ordenação correta dos contactos.

6 Funcionalidades Extra

Embora todas as funcionalidades pedidas no enunciado tenham sido implementadas, o projeto inclui ainda dois elementos adicionais que não eram obrigatórios, mas que melhoram significamente a usabilidade e a experiência do utilizador.

6.1 Persistência de Dados com Ficheiros

Foi implementado um sistema de leitura e escrita em ficheiros para garantir que os dados não se perdem entre sessões

Os contactos são guardados e carregados a partir do ficheiro "CONTACTOS.txt", enquanto o histórico de chamadas é mantido em "HISTORICO.txt".

Este comportamento não era exigido no enunciado, mas permite ao utilizador continuar a usar o programa sem perder os dados anteriormente inseridos.

6.2 Interface com Cores no Terminal

Para tornar a interface mais apelativa e legível, foi criada a biblioteca "cores.h"com macros ANSI que permitem imprimir o texto colorido no terminal.

As cores são usadas para destacar menus, títulos, mensagens de erro e dados importantes melhorando a experiência do utilizador e a clareza das interações.

7 Compilação e Execução

7.1 Requisitos

Para compilar e executar o projeto, é necessário:

- Um compilador C (ex: gcc);
- Um sistema operativo compatível com ANSI espace codes para cores (Linux, macOS ou Windows com terminal compatível);
- Os ficheiros de cabeçalho e implementação organizados num único diretório.

7.2 Compilação

O projeto pode ser compilado com o seguinte comando:

gcc ∗.c −o programa

7.3 Execução

Depois de compilado, o programa é executado com:

./programa

7.4 Ficheiros Gerados

O ficheiro "CONTACTOS.txt"contém todos os contactos, guardados com separador ";". Já o ficheiro "HISTORICO.txt"contém todas as chamadas com os respetivos campos: chamador, destinatário, número e duração.

7.5 Notas

O ficheiro "cores.h"utiliza códigos ANSI, como já referido, que funcionam corretamente na maioria dos terminais modernos. No entanto, em ambientes Windows mais antigos pode ser necessário ativar suporte a ANSI ou executar no Git Bash/WSL.

O tamanho da tabela de hash (1009) está definido diretamente no ficheiro "contacto.h"e apesar de ser um ótimo tamanho e praticamente ilimitado devido ao tipo de estrutura, pode ser ajustado, se necessário, para aplicações maiores.

8 Conclusão

O desenvolvimento deste projeto permitiu-nos consolidar os conhecimentos adquiridos sobre estruturas de dados, modularização de código em C e organização de um sistema complexo com múltiplas funcionalidades.

A aplicação criada cumpre todos os requisitos definidos no enunciado, incluindo a gestão de contactos, a pesquisa por nome, a identificação de chamadas e o histórico navegável. Além disso, foram implementadas funcionalidades extra, como a gravação automática de dados em ficheiro e uma interface com cores, que contribuíram para uma melhor experiência de utilização

Durante o projeto foram enfrentadas e superadas diversas dificuldades técnicas, nomeadamente a gestão de várias estruturas de dados em simultâneo, a normalização de texto para pesquisa, e a implementação da navegação circular num histórico com lista duplamente ligada. Essas dificuldades tornaram-se oportunidades de aprendizagem e de melhoria contínua do código.

Em resumo, o projeto resultou numa aplicação funcional, modular e estável, que aplica de forma prática os conceitos de estruturas de dados e programação estruturada em linguagem C.